Speed Up Matrix Multiplication

Dylan Mumm

1. Abstract

Using various techniques of loop optimizations on a matrix multiplication algorithm in comparison to a non-optimized, naïve implementation of said algorithm, I have set out to see what works best to maximize optimization. Loop interchanging alone resulted in a speedup of 1.27, combining that with loop interchanging increased the speedup to 1.32, and adding loop blocking significantly increased speedup to 3.4.

2. Related work

While the former two algorithms I created by myself from previous knowledge, I needed to resort to researching loop blocking and found Wikipedia's explanation suitable [1]

3. Methodology

Firstly, to improve spatial locality, loop interchange was added onto the naïve approach. Loop unrolling was then added to optimize the algorithm. To go into specifics, the innermost loop was unrolled by a factor of two, insignificantly changing the average speedup. Finally, loop blocking was combined with the previous techniques, more than doubling the speedup to 3.3

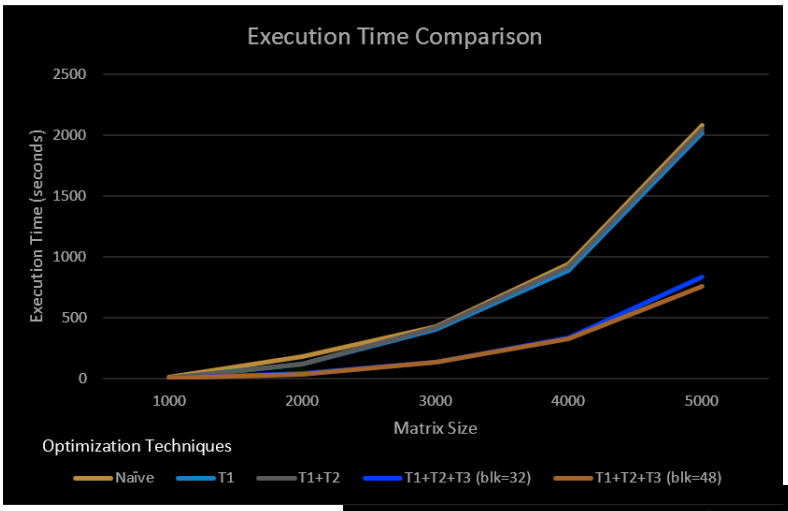
4. Results

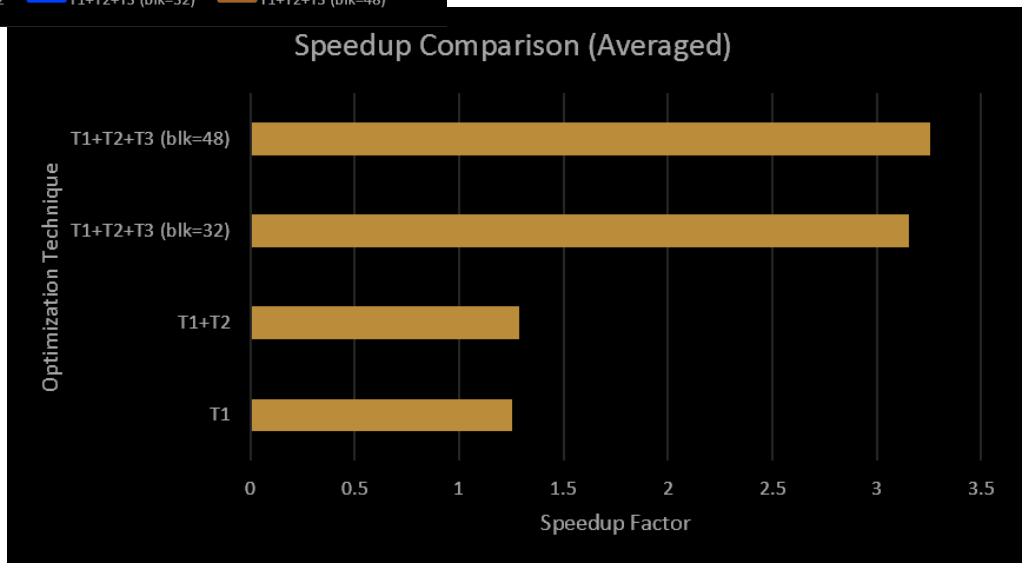
Measurements recorded on an Ubuntu machine with an Intel Core i7 quad-core multithreaded CPU clocked at 3.40 GHz with 16 GB of Memory.

To measure speedup of functions, time of day is received before and after the algorithm, and the difference between the times is calculated. If the standard time function was used, total execution time would have been returned which would not have been restricted to the elapsed time of the algorithm itself.

T1 refers to loop interchanging, T2 to loop unrolling, T3 to loop blocking

Technique / Matrix Size	1000	2000	3000	4000	5000
Naïve + (-O0) ($Time_{naive}$)	16.80	181.33	432.35	961.32	**2251.1**
Transformed code with T1 + (-O0) ($Time_{opt}$)	10.35	121.81	410.69	901.15	**1998.32**
$Time_{naive}/Time_{opt}$	1.62	1.49	1.05	1.06	**1.13**
Transformed code with T1 & T2 + (-O0) ($Time_{opt}$)	9.12	120.05	409.12	898.55	**1997.0**
$Time_{naive}/Time_{opt}$	1.84	1.51	1.06	1.07	**1.13**
Transformed code with T1 & T2 & T3 (blk=32) + (-O0) ($Time_{opt}$)	5.41	41.93	140.33	330.13	**830.69**
$Time_{naive}/Time_{opt}$	3.11	4.32	3.08	2.91	**2.71**
Transformed code with T1 & T2 & T3 & T4 (blk=48) + (-O0) ($Time_{opt}$)	5.05	40.44	133.40	322.35	**760.77**
$Time_{naive}/Time_{opt}$	**3.33**	**4.49**	**3.24**	**2.98**	**2.96**





5. References

[1] “Loop nest optimization,” Wikipedia, 03-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/Loop_nest_optimization. [Accessed: 26-Apr-2020].